MAIS 202 Project Deliverable 2

Yuliya Shpunarska (yuliya.shpunarska@gmail.com)

October 17, 2020

My current best version for implementing neural style transfer can be found in the file neural-transfer-pytorch.ipynb or by clicking here. My model is heavily based off of Alexis Jacq's implementation [1] and Greg Surma's implementation [2], with references to Gregor Koehler's implementation as well [3].

1. Problem Statement

The goal of this project is to create a style transfer model that can take a style image and a content image as inputs, and output an image with the same content but a new style. I will use the pre-trained Keras VGG16 model (and may also implement this with VGG19 later to compare their performance).

2. Data Preprocessing

I will keep the two main datasets that I discussed in the previous deliverable (Best Artworks of All Time and Google Scraped Images). Because this type of algorithm does not need many images (and does not need any tags or metadata on each image), I may reduce the sizes of the datasets either manually or randomly. I will need to do some readings to figure out if it is possible to use several style images (e.g. several paintings from the same artist) to better emulate the style of the person behind them on one content image.

The VGG16 model requires images to be resized to be 224 by 224 pixels and to have 3 channels (Red, Green, Blue). I do not believe this to be a problem since the model comes with a prebuilt pre-processing module, or I can implement it myself using Numpy and Scikit-image. One of the blog posts [2] that I am using as a guide simply resized the images used to a square, then shrunk them down to 224 x 224 px, which seems to be a good option for now.

3. Machine learning model

Discussion of framework

All implementations that I referenced used the pre-trained VGG16 or VGG19 model with a CNN, as was done by Gatys et al. [4] originally. They also used the L-BFGS algorithm to perform gradient descent. A much more detailed discussion can be found in the notebook linked above.

VGG16 and VGG19 have respectively 16 and 19 "weight" layers, as seen in figure 1.

4. Preliminary results

For now, I copied and modified the Alexis Jacq implementation. I intend on keeping the logic of the model similar, except for the normalization section, which I am still working on. The expected result from the tutorial is shown below in figure 2.

In my attempts to make the normalization section more intuitive, I obtained a few unsuccessful results, shown in figures 3 and 4.

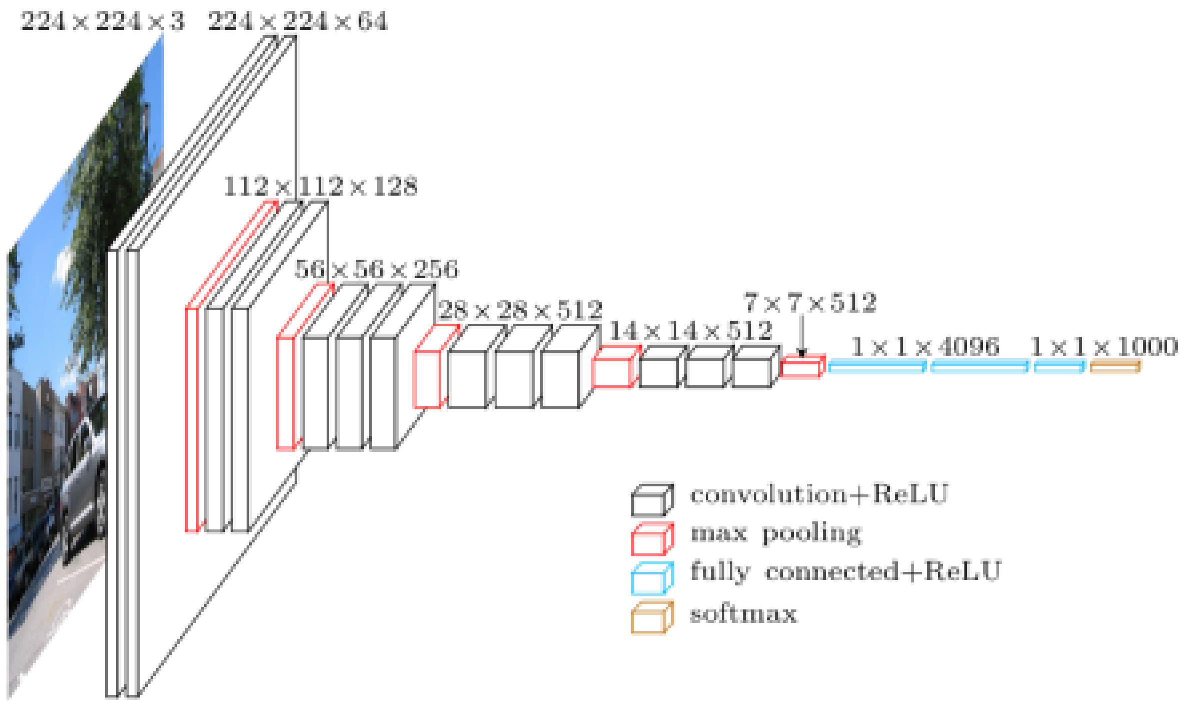


Figure 1: Architecture of VGG16, from http://www.cs.toronto.edu/ frossard/post/vgg16/. Only the section on the left, the feature extraction part of the model (in black and red), is used for style transfer. The rest is useful for classification.

I hope that by learning more about CNNs and PyTorch in the next few weeks, I will be able to modify and update the code in a way that is:

- More intuitive (although potentially at the expense of robustness);
- Capable of taking style inputs from more than one style image (i.e. multiple paintings from one artist).

5. Next steps

As mentioned above, I will modify the code more heavily to focus on the normalization aspect of the preprocessing, and organize it in a way that is easier to understand. I will also need to read about VGG16 to fully understand the functionality of the layers used (among answering many other questions that I wrote down in the notebook).

The more ambitious next step will be to implement a way to merge the style of several images. I do not know if that will actually result in better output pictures, though I have not seen a working implementation for that.

6. Bibliography

[1] Alexis Jacq. *Neural Transfer Using PyTorch.* URL: https://pytorch.org/tutorials/advanced/neural_style_tutorial.html#neural-transfer-using-pytorch. (accessed: 17 October 2020).

[2] George Surma. *Style Transfer - Styling Images with Convolutional Neural Networks.* URL: `https://towardsdatascience.com/style-transfer-styling-images-with-convolutional-neural-networks-7d215b58f461`. (accessed: 16 October 2020).

[3] Gregor Koehler. *Neural Style Transfer in Pytorch.* URL: `https://nextjournal.com/gkoehler/pytorch-neural-style-transfer`. (accessed: 17 October 2020).

[4] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *CoRR* abs/1508.06576 (2015). arXiv: `1508.06576`. URL: `http://arxiv.org/abs/1508.06576`.

```
run [300]:
Style Loss : 2.607505 Content Loss: 9.926151
```

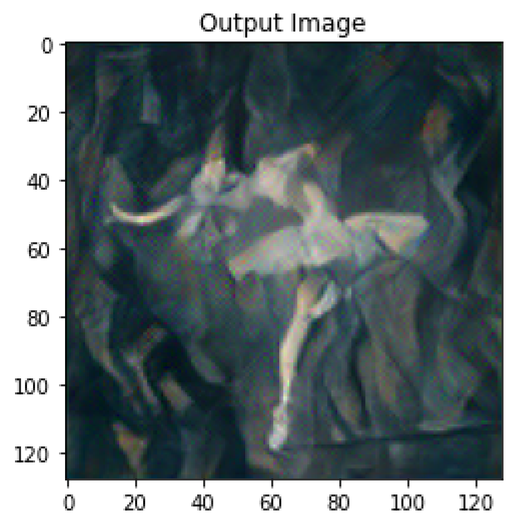
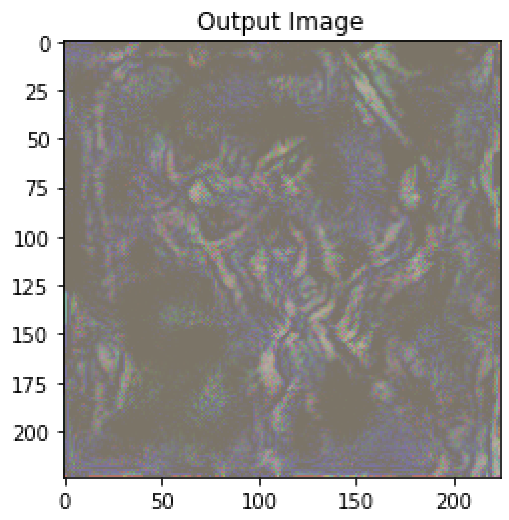


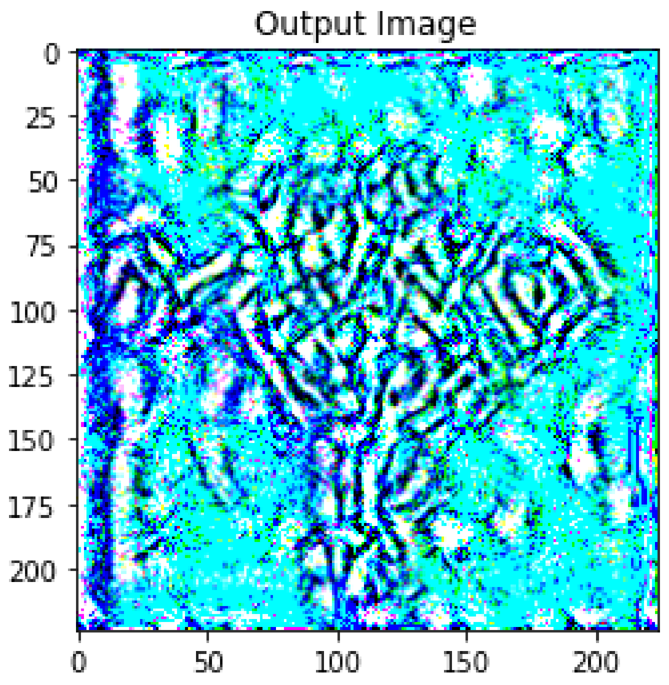
Figure 2: Expected result from the Jacq implementation, which was easy to achieve. Style and content losses for reference.

```
run [300]:
Style Loss : 1046.657104 Content Loss: 9.318633
```



[scale=0.2]

Figure 3: The latest unsuccessful attempt at normalizing the input images in a way that makes sense. Style and content loss for reference, notice that the style loss is much higher than in the successful attempt, but the content losses are comparable.



[scale=0.2]

Figure 4: Another unsuccessful attempt. Both this and the last figure used the same input images as the expected result above.